Implementation von Architekturkonzepten für HW-Agentensysteme

J. Schneider*, M. Naggatz, T. Schmutzler und Rainer G. Spallek

Institut für Technische Informatik
Technische Universität Dresden
D-01062 Dresden

*jsch@ite.inf.tu-dresden.de

Kurzfassung

Anforderungen an zukünftige diensterbringende Systeme sind vor allem Flexibilität, Anpassungsfähigkeit und Ausfallsicherheit. Eine aus der Software- *(SW)* Technologie bekannte Technik zur Umsetzung solcher Anforderungen für diensterbringende Systeme ist die Agenten-Technologie. Ziel des Beitrages ist es, Eigenschaften der SW-Agentensysteme in Hardware *(HW)* umzusetzen. Für die als HW-Agentensysteme bezeichneten diensterbringenden Systeme werden Architekturkonzepte, deren Bereitstellung und Implementationsvarianten beschrieben.

1 Einleitung und Motivation

HW-Agentensysteme weisen entsprechend ihrer Vorbilder [1],[3] aus der SW-Technologie Eigenschaften, wie eine von Benutzereingriffen weitestgehende unabhängige Arbeitsweise *(Autonomie)*, das Auslösen von Aktionen aufgrund eigener Initiativen *(Proaktivität)*, Reaktionen auf Änderungen der Umgebung *(Reaktivität)* und die Kommunikation mit anderen diensterbringenden Systemen *(Interaktivität)* auf. Zusätzliche Eigenschaften sind die Fähigkeit aufgrund zuvor getätigter Entscheidungen bzw. Beobachtungen zu lernen *(Intelligenz)* oder eine Arbeitsweise im Agentenverbund als Multi-Agent-Systeme *(MAS)*. Weitere Klassifikationsmerkmale für Agentensysteme sind in [1],[3],[4] angegeben.

Für eine Umsetzung der aus der SW-Technologie bekannten Agenteneigenschaften in HW wurde in [4] ein Architekturkonzept für HW-Agentensysteme vorgestellt. Abbildung 1 zeigt dieses Konzept einer modularen HW-Agentenarchitektur. Das für HW-Agentensysteme geforderte Optimierungsziel, ein flexibles, anpassungsfähiges und ausfallsicheres System mit möglichst wenig HW-Aufwand zur Verfügung zu stellen, setzt eine leistungsfähige, modulare und universelle Agentenarchitektur voraus. Zur Umsetzung spezifischer Teile einer solchen Agentenarchitektur eignen sich vor allem FPGAs *(Field Programmable Gate Array)*, da diese die gewünschte Flexibilität und Leistungsfähigkeit für eine Implementation besitzen. Diese FPGA-Architekturen, in Abbildung 1 als programmierbare Logik *(PL)* bezeichnet, nehmen zur Ausführung der Agentenfunktionen wie z.B. die Verarbeitung von Ein- und Ausgaben, Kommunikation, Diagnosefunktionen und Aufgabenverteilung speziell angepaßte Funktionseinheiten oder universelle Prozessorkerne auf. Um eine gezielte Beeinflussung der FPGA-Konfiguration und damit der Anpassung des HW-Agentensystems zu erreichen, wurden in [4] und [5] speziell angepaßte Methoden vorgestellt. Diese Ansätze tragen wesentlich zur Umsetzung der Selbstorganisations- und Robustheitsanforderungen des HW-Agenten bei.

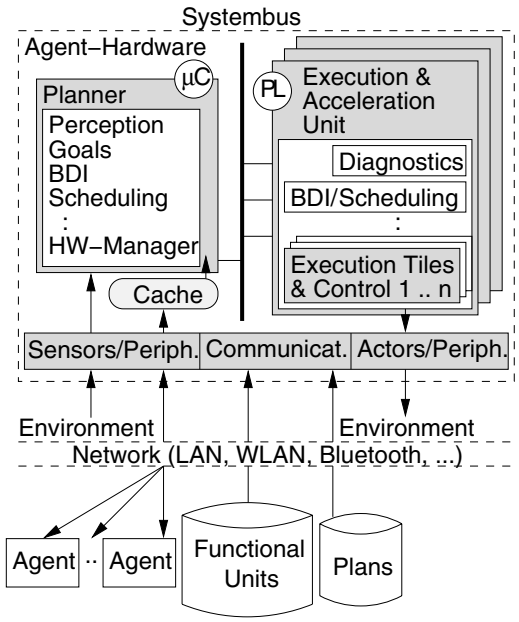

Abbildung 1: modulares Agentenkonzept

Die weiteren Abschnitte des Beitrages sind wie folgt organisiert: Abschnitt 2 stellt Möglichkeiten zur Aufgabenverteilung und Strukturierung von HW-Agentensystemen vor. Abschnitt 3 beschreibt Implementationsvarianten von Architekturkonzepten. Eine Konzeption eines Service-Environments für HW-Agentensysteme zur Überwachung, Steuerung, Datenhaltung und Strukturierung von HW-Agentensystemen wird in Abschnitt 4 vorgestellt. Einige Schlußfolgerungen werden im Abschnitt 5 diskutiert.

2 Plattform für HW-Agentensysteme

Die in Abbildung 1 dargestellte programmierbare Logik-Einheit *(PL)* dient wesentlich zur flexiblen Anpassung der Agentenarchitektur für die Umsetzung des Agentenverhaltens und der Aufgabenverteilung. Dabei existieren zur Realsierung der Agentenstruktur und Umsetzung der Aufgaben vielfältige Möglichkeiten [4].

2.1 Aufgabenverteilung und Strukturierung

In [1],[3],[4] wird eine Zusammenfassung der vielfältigen Arbeitsabläufe in Agentensystemen gegeben. Zu diesen Arbeitsaufgaben gehören u.a. der Informationsaustausch mit Menschen, der Systemumgebung oder anderen Agenten über die Netzwerkschnittstelle, Koordinationsmanagement und Aktivitätsmanagement, Umsetzung aktorischer und kognitiver Fähigkeiten, Fähigkeiten zur Planung und Planausführung und Möglichkeiten zur Überwachung und Selbstdiagnose.

Die Abarbeitung der im Agentensystem geforderten Aufgaben erfolgt durch effizient arbeitende HW- und SW-Lösungen auf parallel angeordneten Systemeinheiten. Die Anordnung der Systemeinheiten kann in unterschiedlichen gleichberechtigten nebeneinanderliegenden funktionalen Einheiten als „horizontale Agentenarchitektur“ oder aber in verschiedenen übereinanderliegenden Schichten (Layers) mit unterschiedlichem Abstraktionsgrad als „geschichtete Architektur“ organisiert werden. Mischformen beider Architekturansätze sind für die Umsetzung von Agentensystemen möglich [4].

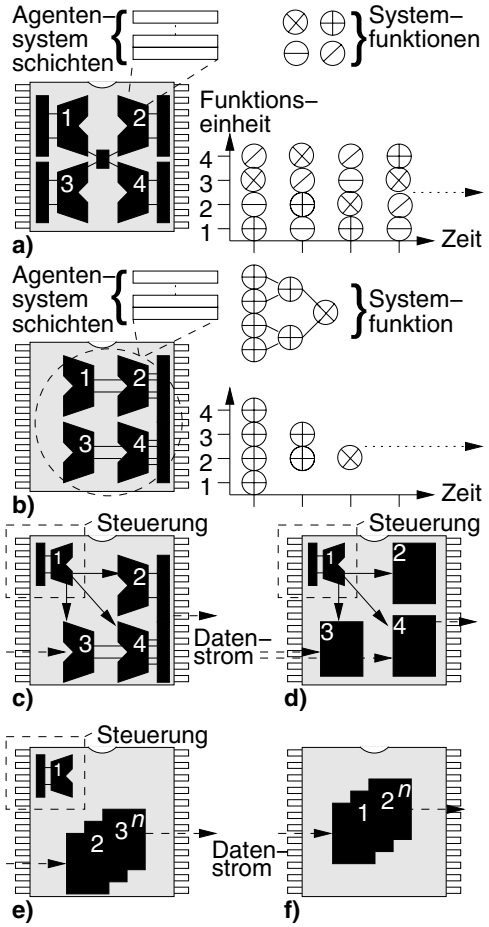

Abbildung 2: Konzepte für Agentenarchitekturen

2.2 Konzepte für Agentenarchitekturen

Für die Umsetzung der Agentenarchitektur auf der programmierbaren Logik-Einheit *(PL)* des HW-Agentensystems sind sowohl spezielle Verarbeitungseinheiten mit spezifisch festgelegter Funktion oder aber universelle Recheneinheiten wie beispielsweise Prozessorkerne nutzbar. Entsprechend der Anforderungen an den HW-Agenten soll eine solche Agentenarchitektur zusammengestellt werden können und nach Konfiguration der programmierbaren Logik-Einheit für Steuerungs- und Datenverarbeitungsaufgaben nutzbar sein [4],[5].

Ziel ist es, die Umsetzung von Agentenfunktionalität und Selbstdiagnosefunktionalität unter Verwendung von Prozessorkernen zu erreichen. Abbildung 2 zeigt verschiedene Möglichkeiten für unterschiedliche Konfigurationsvarianten von Agentenarchitekturen, wobei die Bearbeitung der unterschiedlichen Funktionalität symbolisch durch $\oplus, \ominus, \otimes, \oslash$ dargestellt ist .

Durch Abbildung 2a wird ein Ansatz dargestellt, in welchem die Agentenfunktionalität durch einzelne voneinander unabhängig arbeitende Prozessorkerne realisiert wird. Die Bearbeitung einer spezifischen Agentenfunktion durch mehrere Prozessorkerne ist in Abbildung 2b dargestellt. Der Lösungsansatz in Abbildung 2c spaltet einen Prozessorkern für Steuerungsaufgaben bzw. Diagnoseaufgaben ab. Hierdurch können beispielsweise Überwachungsfunktion und Selbstdiagnosefunktionalität bzw. das Management der Aufgabenverteilung für die einzelnen Prozessorkerne realisiert werden. Der Lösungsansatz in Abbildung 2d ähnelt dem Lösungsansatz in 2c. Hier werden allerdings aufgabenspezifische Funktionseinheiten gesteuert bzw. mit Datenströmen versorgt. Ein Beispiel für eine parallele Realisierung von Steuerfunktionalität und Datenstromverarbeitung durch Integration aufgabenspezifischer Funktionseinheiten wird in Abbildung 2e angegeben. Prozessorkern und aufgabenspezifische Funktionseinheiten arbeiten hier autonom. Eine reine Datenstromverarbeitung durch aufgabenspezifische Funktionseinheiten zeigt Abbildung 2f.

Zur nachfolgenden weiteren Betrachtung in diesem Beitrag ist vor allem der Fall in Abbildung 2a für eine Implementation interessant. Als Grundlage zur Umsetzung dieser Architektur dient eine leistungsfähige RISC-Microprozessor Architektur, welche an die 8-Bit Atmel AVR-Microprozessor Architektur angelehnt ist. Basierend auf einer modularen Konzeption lassen sich verschiedene Agentenarchitekturen individuell zusammenstellen und generativ erzeugen.

3 Implementation von HW-Agenten

Ein Ansatz, möglichst flexible HW-Agentensysteme mit autonomen und selbstregulierenden Eigenschaften zu gestalten, ist die Bereitstellung und Umsetzung vorher genannter HW-Agentenarchitekturen unter Verwendung programmierbarer Prozessorkerne.

3.1 Architekturmerkmale

Basis der vorgestellten HW-Agentenarchitekturen bildet ein Prozessorkern *A8M* mit einer Verarbeitungsbreite von 8 Bit, wobei die Befehlssatzarchitektur der des Atmel ATmega8/16/32 Microprozessors entspricht [6]. Das Konzept dieses Microprozessors wurde modular ausgelegt, so daß es möglich ist, die benötigten HW-Agentenarchitekturen genau den gewünschten Erfordernissen anzupassen.

Eine minimale Implementation eines solchen Prozessorkerns besteht aus Steuerwerk, Rechenwerk, Befehlsdekoder, internem Speicher/Registerfile und einem Ein- und Ausgabeport B. Alle weiteren Komponenten sind optional entsprechend der gegebenen Anforderungen zuschaltbar.

Merkmale eines voll ausgestatteten *A8M* sind eine Taktgeschwindigkeit von 64 MHz, flexible Festlegung von bis zu 6 I/O-Ports, Watchdog-Timer, 8 Bit Timer/Counter, UART, Interrupt-Einheit, 8 Bit ALU mit HW-Multiplizierer, Registerfile mit 32 universell nutzbaren Arbeitsregistern, 64 I/O-Register und 1952x8 Bit internem Speicher. Zusätzlich wurden Signale eingeführt, um eine interne Fehlerüberwachung durchführen zu können. An diese Signale sind Komponenten zur Fehlersuche und Überwachung anzuschließen, welche die Grundlage von Selbstdiagnosefunktionen des HW-Agentensystems bilden. Abbildung 3 zeigt das Blockschaltbild eines vollständig ausgestatteten *A8M*-Microprozessorkerns.

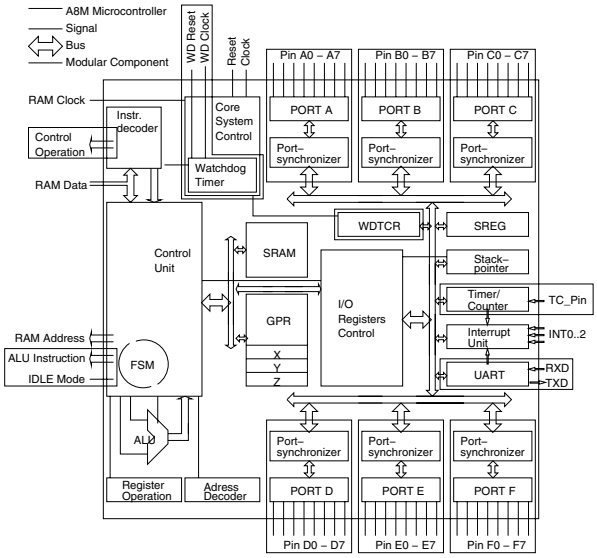

Abbildung 3: A8M-Architektur

Nachfolgende Tabelle 1 zeigt eine Gegenüberstellung von Syntheseergebnissen der beiden HW-Agentenarchitekturvarianten für einen Xilinx Spartan-3 xc3s1000 FPGA.

	min. *A8M*		max. *A8M*	
Logic Utilization	Used	Area	Used	Area
# of Slices	490	6%	733	9%
# of FlipFlops	114	0%	419	2%
# of 4 input LUTs	921	5%	1374	8%
# of bonded IOBs	40	23%	100	57%
# of BRAM	1	4%	1	4%
# of MULT 18x18	1	4%	1	4%
# of GCLKs	1	12%	2	25%

Tabelle 1: Syntheseergebnisse des *A8M*-Kerns

3.2 Generative Bereitstellung

Der modulare Entwurf der HW-Agentenarchitektur erlaubt eine individuelle Zusammenstellung des HW-Agentensystems. Dadurch sind spezielle Systemanpassungen des HW-Agentensystems an die gegebenen Erfordernisse möglich. Um HW-Agentensysteme automatisiert in ihrer Funktion und Struktur anzupassen oder entsprechend zu kombinieren wurde eine Generator-Software entwickelt. Abbildung 4 verdeutlicht die Arbeitsweise dieser Software. Durch diese Generator-SW

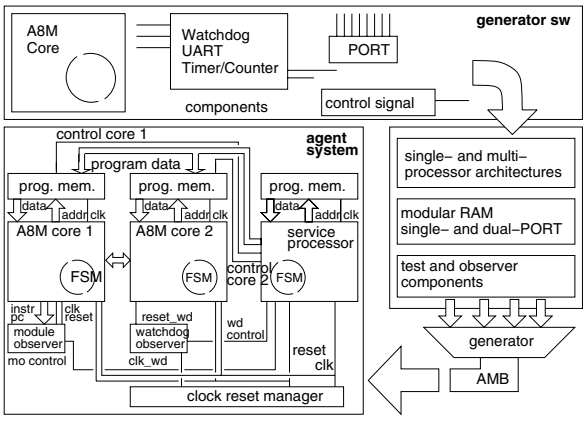

Abbildung 4: Generierung der A8M-Architektur

lassen sich die in das HW-Agentensystem zu integrierenden Komponenten auswählen und anschließend zusammenschalten. Funktionen wie z.B. die Aktivierung interner Testsignale kann ein- bzw. ausgeschalten werden.

Grundlage der Generator-SW bildet die eigens entwickelte Scriptssprache *AMB (Agent Model Builder)*, welche neben der modularen Anpassung der Entwürfe auch eine Verarbeitung des Intel HEX-Formates zum Befüllen der Programmspeicher der Microprozessorkerne und das Anzeigen der hierarchische Abhängigkeiten des Entwurfes der HW-Agentenarchitektur ermöglicht.

3.3 Selbstdiagnosefunktionen

Um HW-Agentensysteme gegenüber Fehlern und Systemausfällen robuster zu gestalten ist es möglich, in die HW-Agentenarchitekturen Test- und Überwachungskomponenten bei Bedarf einzubinden. Diese Test- und Überwachungskomponenten ermöglichen eine Selbstüberwachung und die dadurch bestehende Möglichkeit zur Einleitung der Selbstreperatur des HW-Agentensystems. Für den Anschluß dieser Komponenten wurden aus dem Prozessorkern spezielle Test- und Diagnosesignale herausgeführt. Durch diese Umsetzungsvariante verläuft die Fehlerüberwachung und Diagnose parallel zu den für die Erfüllung der Agentenfunktionen verwendeten Microprozessorkernen. Die Testkomponenten realisieren eine Fehlervorverarbeitung und stehen in direkter Verbindung mit den Serviceprozessoren. Die evtl. auftretenden Fehlerergebnisse werden an diese weitergereicht. Vorteil dieser Verfahrensweise ist die durch die Testkomponenten realisierte Zwischenschicht (Wrapper) einer abstrakten Fehlerbehandlung für die Serviceprozessoren. Abbildung 5 verdeutlicht das Konzept zur Selbstdiagnose in HW-Agentensystemen, welches aus einem *A8M*-Serviceprozessor und vier *A8M*-Prozessorkernen mit angeschlossenen, von einander verschiedenartigen Test- und Überwachungskomponenten besteht.

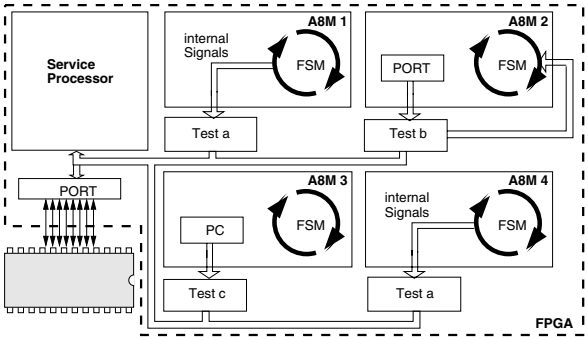

Abbildung 5: Prinzip der Selbstdiagnose

Serviceprozessoren

Die Serviceprozessoren sind zentraler Bestandteil des HW-Agentensystems und werden zur Steuerung und Selbstdiagnose eingesetzt. Vorzugsweise kommen für Serviceprozessoren die minimierten *A8M*-Prozessorkerne zur Anwendung. Sie können einzeln oder mehrfach in den HW-Agenten eingebunden werden. Eine redundante Auslegung der Serviceprozessoren im HW-Agentensystem macht Sinn und vermindert die Wahrscheinlichkeit von Ausfällen des HW-Agentensystems. Abbildung 6 zeigt ein Konzept zur Realisierung zweier sich selbst überwachender Serviceprozessoren.

Reagiert ein HW-Agentensystem aufgrund schwerwiegender Systemfehler nicht mehr, dann dienen Serviceprozessoren als wesentliches Hilfsmittel für die

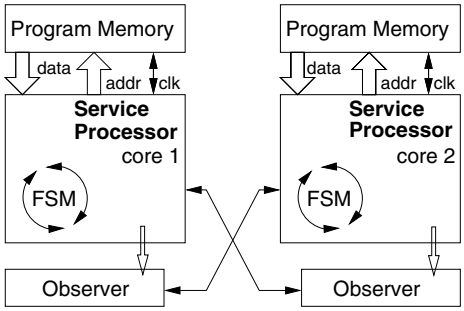

Abbildung 6: Diagnose von Serviceprozessoren

Fehlerbehandlung. Diese initiieren ohne Eingriff des Benutzers die Abarbeitung einer Fehlerroutine. Die eigentliche Fehlerroutine wird als SW abgearbeitet und kann somit für verschiedene Problemfälle individuell erstellt werden bzw. ist leicht änderbar. Bezüglich der Überwachungsfunktion sind zwei verschiedene Arbeitsmodi möglich. Eine Möglichkeit ist eine vollständige komplett parallele Arbeitsweise der Serviceprozessoren. Die andere Möglichkeit besteht darin, einen der Serviceprozessoren in den IDLE-Mode zu versetzen, während der andere die Systemüberwachung übernimmt. Im Fall eines Ausfalls des überwachenden Serviceprozessors übernimmt der andere die Überwachungs- und Diagnosefunktion des HW-Agentensystems. Serviceprozessoren arbeiten unabhängig von den *A8M*-Microprozessoren zur Ausführung der Agentenfunktionalität.

Neben der Realisierung der Überwachungs- und Diagnosefunktionen werden die Serviceprozessoren auch noch dazu genutzt, Aufgaben an die zur Realisierung der Agentenfunktionalität vorhandenen *A8M*-Prozessorkerne zu verteilen. Dazu wird der zur Abarbeitung bestimmte Programmcode in den Programmspeicher des jeweiligen *A8M*-Prozessorkerns durch den Serviceprozessor geschrieben.

Selbsttest: Interrupt Check

Die *A8M*-Architektur besitzt entsprechend seinem Vorbild, der Atmel AVR-Architektur Interrupts und Möglichkeiten zur Interruptbehandlung [6]. Externe Interrupts werden an den Pins INT0, INT1 und INT2 ausgelöst. Durch diese Pins wird eine der Überwachungs- und Diagnosefunktionen des *A8M*-Prozessorkerns realisiert.

Duch Auslösen eines Interrupts wird ein internes Testprogramm mit festgelegtem Ablauf, Ein- und Ausgaben gestartet. Die Selbsttestkomponente „Interrupt Check (IC-Komponente)“ kümmert sich selbstständig um den Start eines Tests und dessen Auswertung. Abbildung 7 zeigt die Integration dieser Selbsttestkomponente in die *A8M*-Architektur.

Die IC-Komponente besitzt einen mit 1 MHz getakteten Timer, welcher bei Ablauf ein Interruptsignal an den *A8M*-Prozessorkern sendet. Durch den Servicepro-

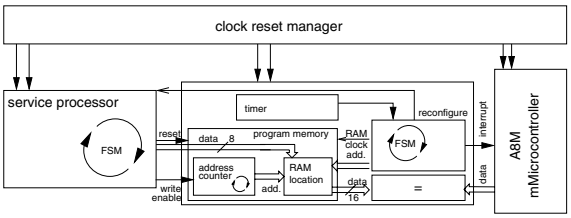

Abbildung 7: Test durch IC-Komponente

zessor wird der in der IC-Komponente enthaltene Vergleichsspeicher mit Werten der zu erwartenden Testergebnisse und des Timerwertes versorgt. Wird ein neuer Test durch die IC-Komponente gestartet, dann wird zuerst der Timerwert gelesen. Sollte der Test des *A8M*-Prozessorkerns länger als die vorgegebene Zeitspanne dauern, dann wird der Serviceprozessor mit einem Fehlerhinweis informiert. Der eigentliche Testablauf läuft so ab, daß die aus dem *A8M*-Prozessorkern kommenden Werte mit den Werten im Vergleichsspeicher der IC-Komponente verglichen werden. Tritt ein Unterschied auf, wird das als Fehler interpretiert und wiederum wird der Serviceprozessor informiert.

Selbsttest: Watchdog Observer

Die im *A8M*-Microprozesserkern implementierte Watchdog-Funktionalität stellt für die Selbsttestkomponente „Watchdog Observer (WO-Komponent)“ ein internes Signal des Watchdog-Reset zur Verfügung. Abbildung 8 zeigt die Integration dieser Selbsttestkomponente in die *A8M*-Architektur.

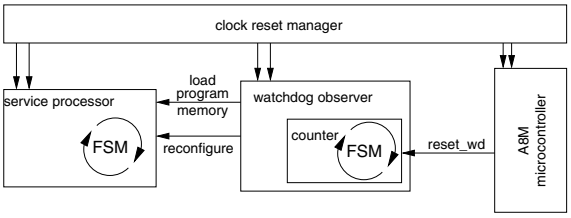

Abbildung 8: Test durch WO-Komponente

Wird vom *A8M*-Microprozesserkern ein Watchdog-Reset veranlaßt, dann folgt ein durch den Watchdog-Timer ausgelöster globaler Reset des *A8M*-Microprozesserkerns. Soll es nicht zum Watchdog-Reset kommen, dann muß der Watchdog-Timer rechtzeitig zurückgesetzt werden. Läuft der Watchdog-Timer ab, wird der globale Reset ausgelöst.

Die WO-Komponente stellt eine Erweiterung der Watchdog-Funktionalität des *A8M*-Microprozesserkerns dar. Die WO-Komponente besitzt einen internen 2 Bit Zähler der die ausgelösten Watchdog-Resets mitzählt. Dieser Zähler bewirkt folgende Ereignisse:

- **2x Watchdog-Reset:** veranlassen, daß der Befehlsspeicher neu geladen wird und es erfolgt nach dem Neuladen ein globaler Reset des *A8M*-Microprozesserkerns durch den Serviceprozessor.
- **4x Watchdog-Reset:** veranlassen, daß das HW-Agentensystem neu konfiguriert wird.

Durch die WO-Komponente lassen sich vor allem Systemfehler eingrenzen, welche auf einen internen Fehler in der Reset-Struktur oder auf einen permaneten Fehler in der Programmstruktur (z.B. Endlosschleife) zurückzuführen sind.

Die Funktionalität dieser WO-Komponente ist durch zusätzliches Auswerten von Zählerereignissen bzw. durch Erweiterung der Bitbreite des Zählers beliebig erweiterbar.

Selbsttest: Module Observer

Eine Überwachung der internen Steuersignale des *A8M*-Microprozesserkerns wird von der Selbsttestkomponente „Module Observer (MO-Komponente)“ vorgenommen. Abbildung 9 zeigt die Integration dieser Selbsttestkomponente in die *A8M*-Architektur.

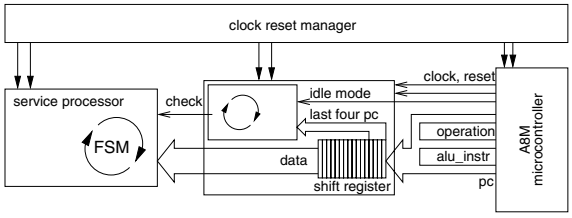

Abbildung 9: Test durch MO-Komponente

Die zu überwachenden Signale sind hierbei der Befehlszähler und die der Steuerwerksbefehle und des Rechenwerks. Diese Werte werden in einem Trace File aufgezeichnet. Es kann 16 Werte aufnehmen und wird dazu genutzt, um den Fehlerzustand vor der Fehlerbehandlung auszulesen. Weiterhin wird von der MO-Komponente kontrolliert, ob die letzten vier Befehlszähler unterschiedlich sind. Sind vier Befehlszähler gleich, dann kann davon ausgegangen werden, daß sich der *A8M*-Microprozesserkern in einer Endlosschleife befindet. Werden vier gleiche Befehlszähler erkannt, dann kann die SW im Serviceprozessor entscheiden, wie der Fehler behandelt wird. Für die Fehlerbehandlung besteht wiederum die Möglichkeit, einen Reset am relevanten *A8M*-Microprozesserkern auszulösen oder aber das HW-Agentensystem neu zu konfigurieren.

3.4 Multikernarchitektur

Entsprechend der Abbildung 2 findet eine Umsetzung der Agentenfunktionalität durch den Einsatz programmierbarer Prozessorkerne statt. Um eine möglichst effektive Arbeitsweise im HW-Agenten zu gewährleisten, ist der parallele Einsatz mehrerer *A8M*-Prozessorkerne möglich.

Um die Verwendung der entwickelten HW-Agentenarchitektur deutlich zu machen, zeigt Abbildung 10 ein Implementationsbeispiel bestehend aus:

- 2x *A8M*-Microprozesserkern mit jeweils einem Dual-Port RAM,
- 1x Serviceprozessor, ebenfalls durch *A8M*-Microprozesserkern realisiert,
- 1x Watchdog-Observer,
- 1x Module-Observer,
- 1x Clock Reset Manager.

Die Anbindung der Selbsttest-Komponenten wurde für dieses Implementationsbeispiel willkürlich gewählt. Eine Einbringung der in Abschitt 3.3 vorgestellten Selbsttestkomponenten pro *A8M*-Microprozesserkern wäre ebenfalls möglich. Zur Erhöhung der Ausfallsicherheit kann entsprechend der Abbildung 6 ein zweiter Serviceprozessor der HW-Agentenarchitektur hinzugefügt werden.

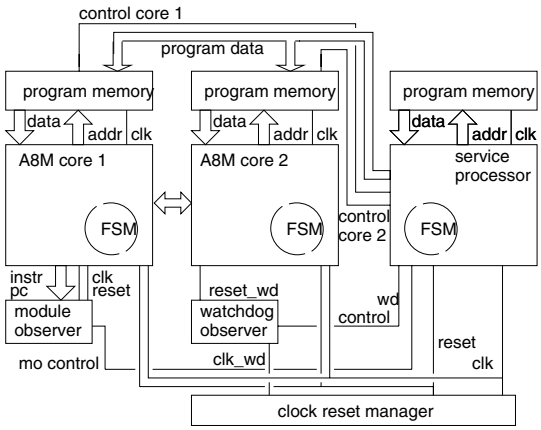

Abbildung 10: HW-Agentenarchitektur

Die Auswertungen der Selbsttestkomponenten werden direkt an den Serviceprozessor gemeldet, welcher ggf. für eine Fehlerbehandlung genutzt werden kann. Weiterhin ist der Serviceprozessor für eine Aufgabenverteilung innerhalb des HW-Agentensystems zuständig. Er lädt die Befehlsspeicher des jeweiligen *A8M*-Microprozesserkerns. Der Clock Reset Manager stellt die benötigten Takte und das Systemreset zur Verfügung.

In Tabelle 2 ist das Syntheseergebnis zur Implementation auf einem Xilinx Spartan-3 xc3s1000 FPGA der in Abbildung 10 dargestellten Agentenarchitektur aufgezeigt. Das implementierte HW-Agentensystem nimmt etwa 21% auf der verwendeten programmierbaren Logik ein. Je nach bestehenden Anforderungen ist bei Verwendung eines Xilinx Spartan-3 xc3s1000 FPGA noch genügend Platz für Erweiterungen durch z.B. Hinzufügen weiterer *A8M*-Microprozesserkerne möglich.

	HW-Agentensystem	
Logic Utilization	Used	Area
# of Slices	1633	21%
# of FlipFlops	531	3%
# of 4 input LUTs	3110	20%
# of bonded IOBs	37	21%
# of BRAM	6	25%
# of MULT 18x18	3	12%
# of GCLKs	2	25%
# of DCMs	1	25%

Tabelle 2: Syntheseergebnisse des HW-Agenten

Wird hingegen ein kleinerer Vertreter der Spartan-3 Familie, der xc3s200 FPGA verwendet, dann wird mit dem in Abbildung 10 dargestellten HW-Agentensystem eine Auslastung der programmierbaren Logik um 85% erreicht. Eine Gegenüberstellung des Ressourcenverbrauchs für den Xilinx Spartan-3 xc3s1000 und xc3s200 FPGA zeigt Abbildung 11. Gegenübergestellt werden die Flächenauslastung bzgl. des minimal und maximal ausgestatteten *A8M*-Microprozesserkerns, Einzel- und Dualprozessorsystems und des HW-Agentensystems entsprechend Abbildung 10. Durch das modulare Kon-

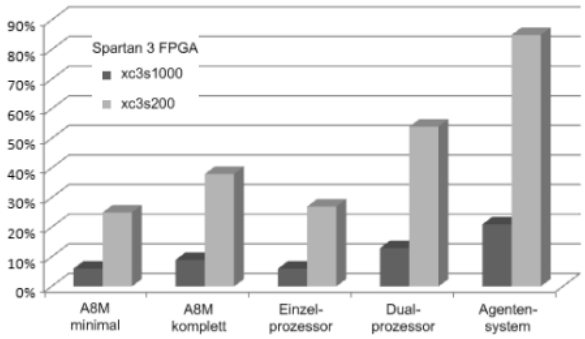

Abbildung 11: Syntheseergebnisse

zept ist eine effiziente Auslastung der programmierbaren Logik möglich und das HW-Agentensystem kann durch die Generator-SW leicht angepaßt werden.

4 Service Environment *„amsys“*

Für Netzwerke, bestehend aus mehreren HW-Agentensystemen, bedarf es einer zentralisierten Verwaltung. Ein solches Managementsystem bewältigt Verwaltungs- und Steuerungsaufgaben und soll die Ausfallsicherheit durch Redundanz erhöhen.

4.1 Systemkonzept

HW-Agentensysteme, welche nach einem Architekturansatz entsprechend der Abbildung 1 implementiert sind, besitzen durch ihren Netzzugang Verbindung zu anderen HW-Agenten bzw. zu ihrer Umwelt. Durch die Möglichkeit der Vernetzung beste-

hen vielfältige Möglichkeiten zur Zusammenarbeit zwischen den HW-Agentensystemen. Für das Management der HW-Agentensysteme und der im Hinblick auf die mit HW-Agentensystemen in Verbindung stehenden vielfältigen Aufgaben wurde eine in der Programmiersprache Java erstellte „Service Environment"-SW *„amsys"* (**A**gent **M**anagement **Sys**tem) entwickelt. Abbildung 12a zeigt das Systemkonzept von *„amsys"*.

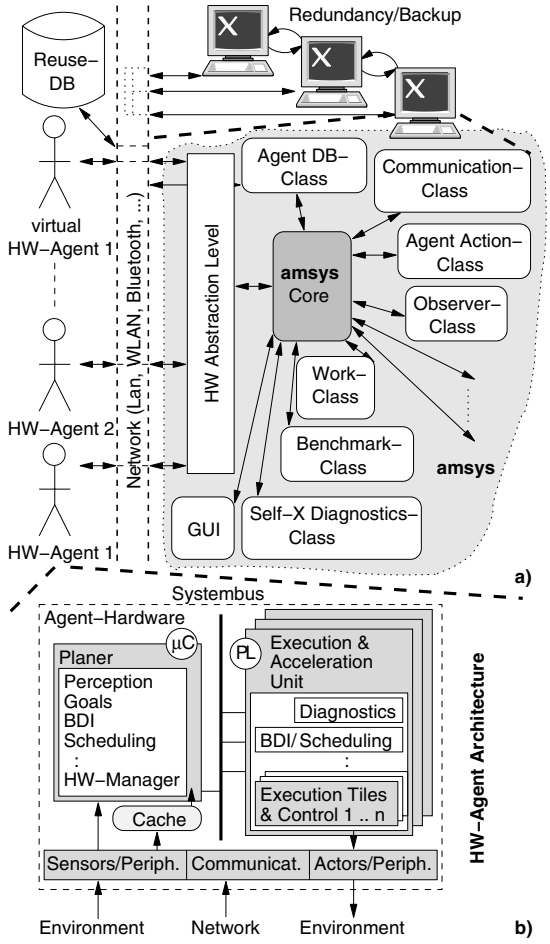

Abbildung 12: Systemkonzept *„amsys"*

4.2 Aufgaben

Zu diesen Aufgaben gehören in erster Linie die Verwaltung von HW-Agentensystemen in dedizierten Agentennetzwerken. Von den im Netzwerk identifizierten HW-Agentensystemen sollten ihr Systemstatus (rechnend, rechenbereit, blockiert) und ihre jeweilige Systemkonfiguration (z.B Anzahl Ausführungseinheiten, Programm- und Datenspeicher, Kapazität und Typ der programmierbaren Logik, ...) bekannt sein. Ausfälle von HW-Agentensystemen sollten frühzeitig bemerkt werden, so daß eine Aufgabenumverteilung auf andere HW-Agentensysteme erfolgen kann.

Um mehreren HW-Agentensystemen eine gemeinsame Aufgabe zur Lösung zu übergeben, ist eine Gruppierung der HW-Agentensysteme mitunter sinnvoll. Durch die „Service Environment"-SW lassen sich solche virtuellen Gruppierungen wie die Bildung von hierarchischen Baumstrukturen (siehe Abbildung 13 a) oder gleichberechtigten HW-Agentenstrukturen (siehe Abbildung 13 b) erreichen.

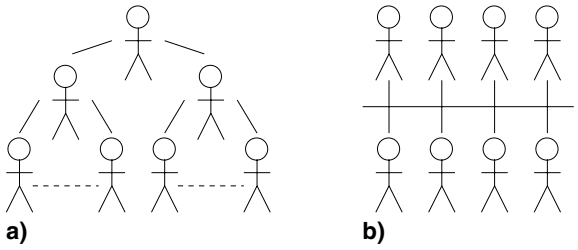

Abbildung 13: Gruppierungen von HW-Agenten

Ein weiterer wichtiger Punkt ist die Kommunikation zwischen den HW-Agentensystemen. Damit eine Koordination dieser Kommunikation erfolgen kann, verläuft diese über die „Service Environment"-SW. Die eigentliche Kommunikation erfolgt über eine einfache Scriptsprache.

Durch eine Datenbankanbindung ist es möglich, die Verwaltung von beispielsweise HW-Agenten-Konfigurationen, Konfigurationsdaten zur Adaption der programmierbaren Logik, SW-Programme zur Abarbeitung auf den Microprozessorkernen und Ausführungsplänen zur Lösung bestimmter Aufgaben vorzunehmen. Auf diese Daten kann durch die „Service Environment"-SW zugegriffen werden und sie sind dadurch für alle HW-Agentensysteme verfügbar.

Die „Service Environment"-SW wird u.a. zur Verwaltung der Konfigurationsdaten und Programmdaten verwendet. Entsprechend der von den HW-Agentensystemen kommenden Anforderungen werden Konfigurationsdaten für die programmierbare Logik bereitgestellt. Nach der Konfiguration von beispielsweise Microprozessorkernen kann vom HW-Agentensystem entspechende SW zur aktuellen Problemlösung angefordert werden.

Neben der Verwaltung und Administration von HW-Agentensystemen existieren zusätzlich virtuelle Agentensysteme, welche das Verhalten realer HW-Agentensysteme nachbilden. Durch den Start mehrerer solcher virtueller Agentensysteme kann die Zusammenarbeit mehrerer Agentensysteme simuliert werden.

Die „Service Environment"-SW stellt eine graphische Benutzerschnittstelle zur Verfügung, über welche Statusabfragen, Informationsausgaben und Nutzereingaben möglich sind.

5 Zusammenfassung und Ausblick

In diesem Beitrag wurde die Implementation von HW-Agentenarchitekturkonzepten beschrieben. Diese basieren auf dem *A8M*-Microprozessorkern, welcher an die Atmel AVR (Atmega8/16/32) Architektur angelehnt ist. Das HW-Agentenarchitekturkonzept kann

SW-basiert generativ bereitgestellt werden und kann als Multi-Kern-System arbeiten. Dabei ist eine Trennung der *A8M*-Microprozessorkern-Funktionalität in Serviceprozessoren und *A8M*-Microprozessorkerne zur Ausführung der Agentenfunktionalität möglich. Zur Erhöhung der Ausfallsicherheit können in den *A8M*-Microprozessorkern zusätzlich unterschiedliche Selbsttestkomponenten implementiert werden. Für das Management von HW-Agentensystemen wurde eine in der Programmiersprache Java erstellte „Service Environment"-SW vorgestellt. Diese wird zur Verwaltung, Überwachung und Organisation von HW-Agentensystemen genutzt.

Zukünftige Arbeiten zielen vor allem auf den praktischen Einsatz der HW-Agentenarchitektur ab. Zusätzlich werden weitere Ansätze entwickelt, welche als Selbsttestkomponenten in den *A8M*-Microprozessorkern implementiert werden können. Weiterhin ist die Vervollständigung des „Service Environment"-SW geplant.

Literatur

[1] Wooldridge, M.: *An Introduction to MultiAgent Systems*, Wiley Publishers, 2002

[2] Müller, J.: *The Design of Intelligent Agents*, Lecture Notes in Artificial Intelligence 1177, 1996

[3] Fricke, St.: *Werkzeuggestützte Entwicklung kooperativer Agenten im Dienstkontext*, Dissertation, TU-Berlin, 1996

[4] J. Schneider, F. Niederlein und Rainer G. Spallek: *Architekturkonzeption und HW-Unterstützung für Agentensysteme*, Dresdner Arbeitstagung Schaltungs- und Systementwurf DASS'06, 10.–11. Mai 2006, Dresden

[5] J. Schneider, F. Niederlein und R. G. Spallek: *Sequencing graph based self-organization for hardware agent systems*, ICYR 2006, September 18-20, 2006, Zielona Gora, Poland

[6] Atmel: *Datasheet: ATmega32 (L)*, http://www.atmel.com/dyn-/resources/prod_documents/doc2503.pdf